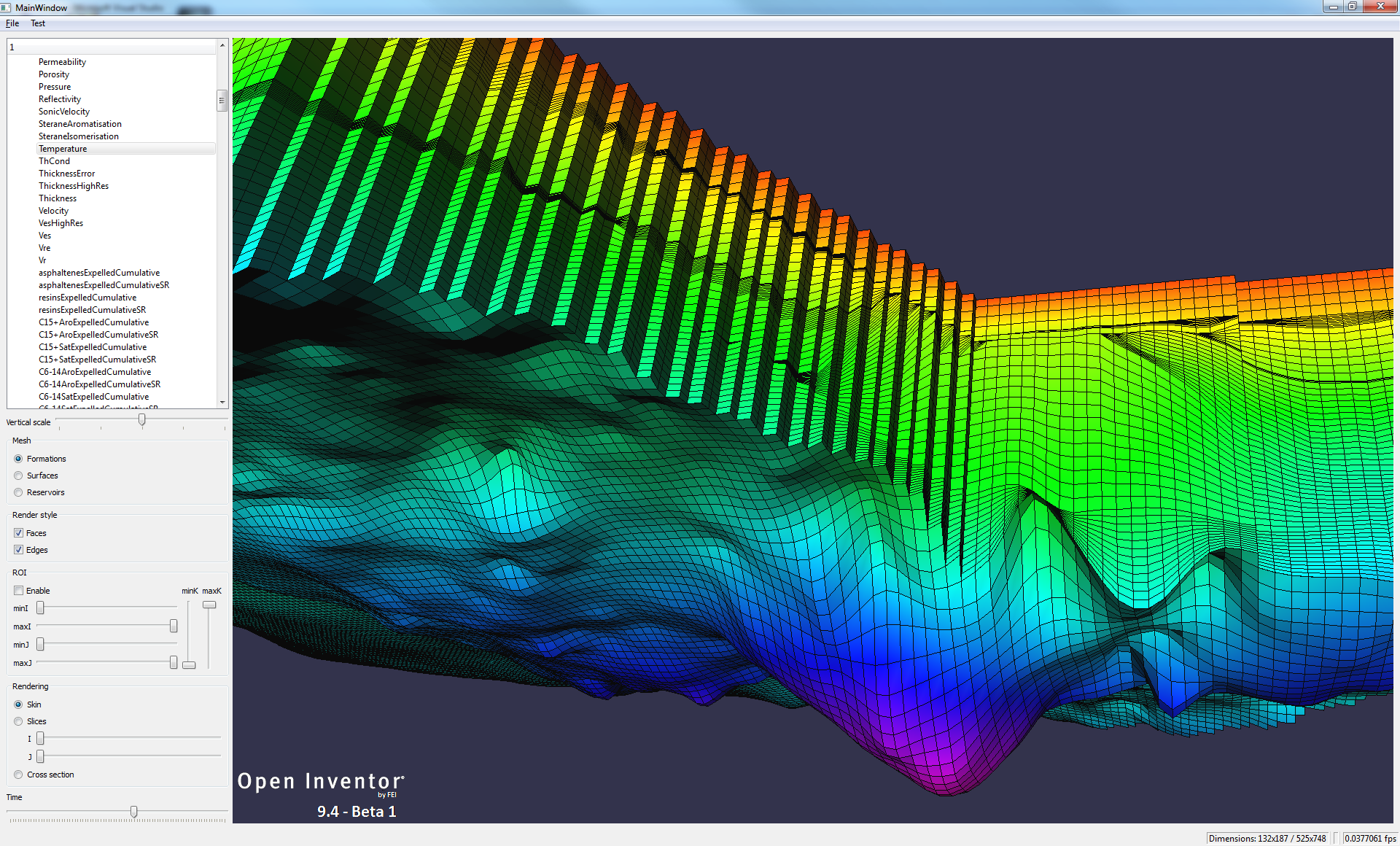
4D prototype performance

*Serge van der Ree, 21-03-2014*

This document describes the current performance of the Basin Modelling 4D prototype, using a large data set. The mentioned target fps (frames per second) values are taken from the document ‘BPA2 4D Web Viewer Evaluation & Requirements’.

# Data

The data set was provided by Kris Hopstaken. It contains formations defined on a low resolution grid, and reservoirs defined on a high resolution grid.



* Nr. Of snapshots (time steps): 84
* Low-resolution grid: 132x187
* High-resolution grid (reservoirs): 525x748
* Size on disk: approx. 36GB

# Skin extraction

In order to view our data, some kind of renderable mesh needs to be extracted, e.g. the outer skin, slices, iso-surfaces etc. The extraction process takes a volume mesh (e.g. hexahedral, tetrahedral) and generates a surface mesh (e.g. triangles) from it.

Preparing the surface mesh for rendering requires a couple of extra steps from Open Inventor, such as generating the vertex buffers for the triangle mesh and the gridline mesh, and generating normal vectors.

OpenGL mesh

Surface mesh

Volume mesh

The extraction process can be done implicitly by Open Inventor when specifying a rendering node for our volume data, but it can also be done explicitly as a pre-processing step, adding only the extracted surface mesh to the scene graph. Doing both allows us to to get an idea of where most of the time is spent.

# Measurements

## Time roaming (target: 10 fps)

### Skin

These measurements are taken by rendering the reservoir meshes once for every timestep, directly after loading the data. The first column show the times for the ‘normal’ way of rendering, i.e. with implicit extraction. The second column shows times when using pre-extracted meshes. The third column does this as well, but here grid lines were not rendered. The last column contains the numbers for the situation where the test has already been performed a number of times, and all the necessary data has been generated and cached.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time per snapshot | Normal | Pre-extracted | Pre-extracted, without grid lines | All generated and cached (no grid) |
| Avg (s) | 2.4 | 1.8 | 0.3 | 0.08 |
| Max (s) | 10.7 | 8.0 | 1.5 | 0.3 |

This table indicates that the generation of the line mesh for edge rendering is by far the most time consuming. In practice this may not be a problem; rendering edges on such a high resolution mesh causes the output to be mostly black, because of the high density of the grid lines. Without edge rendering, roaming through time seems fast enough for interactive use, especially when all the necessary data has been generated and cached.

### Slices

Time roaming using I,J slices is plenty fast: 50+ fps

## Property change (target: 2 fps)

* Local: instantaneous
* Global: 3.5 s

The difference between the two is that in the ‘global’ case, the property is scanned across all time steps to find the global minimum and maximum value, in order to make the color mapping consistent.

## Space roaming (target: 10fps)

Moving I and J slices through the high-resolution data is very fast: 50+ fps

## Camera movement (target: 10+ fps)

In this situation no data needs to be generated, only the viewing transformation are updated. This shows in the rendering performance: 400+ fps (V-sync disabled obviously) for skin rendering.

# Test system

* Intel Xeon E5620 CPU
* 64 GB main memory
* 2x Nvidia Quadro 5000 graphics
* Windows 7 64-bit

# Conclusions

There’s probably some room for improvement in the implementation of the OIV interface to access our data, but since the mesh extraction does not seem to be the bottleneck, this is unlikely to give much of an improvement.

However, even in the current state rendering seems to be sufficiently fast, provided that edge rendering is disabled for the skin mesh. Other rendering modes have no performance issues whatsoever.